Machine Learning

Stanford University
Professor Andrew Ng

Jordan Hong

June 6, 2020

Contents

1 Introduction

1.1 What is Machine Learning

1. Machine Learning
   - Grew out of work in Artificial Intelligence (AI)
   - New capabilities for computers
2. Examples:
   - database mining
   - applications can't programby hand (handwriting recognition, Natural Language Processing (NLP), Computer Vision)
   - Neuromorphic applications
3. Definition
   - Arthur Samuel(1959)

     > Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed.

   - Tom Mitchell(1998)

     > Well-posed Learning Problem: A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

4. Machine Learning in this course:
   (a) Suupervised Learning
   (b) Unsupervised Learning
   (c) Others: reinforcement learning, recommender systems
   (d) Practical application techniques

1.2 Supervised Learning

In supervised learning, the *the right answer* is given. For example:

1. Regression: predict real-valued output.
2. Classification: predict discrete-valued output.

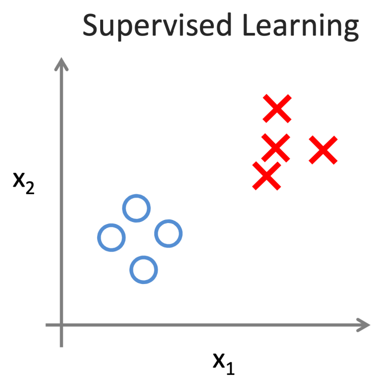

Figure 1: Supervised Learning

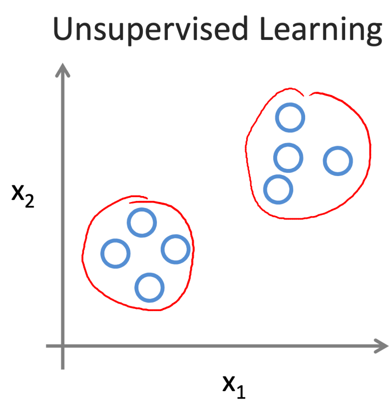

Figure 2: Unsupervised learning

1.3 Unsupervised Learning

The right answer is not given, e.g. cocktail problem (distinguishing two voices from an audio file.)

2 Linear Regression with One Variable

2.1 Model Representation

2.1.1 Notations

For a training set:

- **m** = Number of training examples.
- **x** = “input” variable / features.

- **y** = "output" variables / "target" variable.
- **(x,y)** - one training example.
- **($\mathbf{x^i}$,$\mathbf{y^i}$)** denotes the i^{th} training example

2.1.2 Hypothesis Function

A hypothesis function (h) maps input (x) to estimated output (y). How do we represent h?

$$\boxed{\textbf{Hypothesis Function} \quad h_\theta(x) = \theta_0 + \theta_1 x} \tag{1}$$

We can apply *Univariate linear regression* with respect to x.

2.2 Cost Function

Recall 1. The θ_is are parameters we have to choose. The intuition is is that we want to choose θ_i s such that h_θ is closest to y for our training examples (x,y).

$$\boxed{\textbf{Cost Function} \quad J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2} \tag{2}$$

Summary

1. **Hypothesis** $h_\theta(x) = \theta_0 + \theta_1 x$
2. **Parameters** θ_0, θ_1
3. **Cost Function** $J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$
4. **Goal** $\min_{\theta_0,\theta_1} J(\theta_0, \theta_1)$

2.3 Gradient Descent

2.3.1 Intuition

1. We have some function $J(\theta_0, \theta_1)$, we want to $\min_{\theta_0,\theta_1} J(\theta_0, \theta_1)$
2. Outline: start with some θ_0, θ_1, keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we end up at a minimum.

2.3.2 Gradient Descent Algorithm

Algorithm
repeat until convergence{

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta_0, \theta_1) \quad \text{(for j=0 and j=1).}$$

}

Notes

1. the := denotes non-blocking assignment, i.e. simultaneously updates $\theta_0 and \theta_1$
2. We use the derivative to find a local minimum.
3. α denotes the learning rate. Gradient descent can converge to a local minimum even when the learning rate α is fixed. As we approach a local minimum, gradient descent will automatically take smaller steps. Therefore it is not needed to decrease α over time.

2.3.3 Gradient Descent with Linear Regression

Recall, we have:

1. Gradient Descent Algorithm:

   repeat until convergence{

   $$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad \text{(for j=0 and j=1).}$$

   }

2. Linear Regression Model:

   $$h_\theta(x) = \theta_0 + \theta_1 x$$

   $$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

We can substitute the above equations, which gives us:

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

3 Review of Linear Algebra

This is section is a basic review of linear algebra. I have skipped this section for now and will come back to it if time permits.

4 Linear Regression with Multiple Variables

4.1 Multiple features

Recall in the single variable case, we have a single input (x), two parameters(θ_0, θ_1). The hypothesis can be expressed as:

$$h_\theta(x) = \theta_0 + \theta_1 x.$$

Now, consider a generalized case where there are multiple features: X_1, X_2, X_3. The information can be organized in a table with example numerical values:

Sample Number (i)	X_1	X_2	y
1	6	87837	787
2	7	78	5415
3	545	778	7507
4	545	18744	7560
5	88	788	6344

Table 1: Sample Table

From Table 1, one can see that each row is a sample a feature on each column.

4.1.1 Notation

1. **n**: number of features.
2. $\mathbf{x^{(i)}}$: (row vector) input features of the i^{th} training example. i= 1, 2,. . . , m.
3. $\mathbf{x^{(i)}}_\mathbf{j}$: value of feature j in the i^{th} training example. j= 1, 2, . . . , n.

4.1.2 Hypothesis

Previously,

$$h_\theta(x) = \theta_0 + \theta_1 \cdot x$$

Now, we can extend the hypothesis to :

$$h_\theta(x) = \theta_0 \cdot 1 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2$$

For convenience of notation, let's define x_0=1, i.e. $x^i{}_0$=1 $\forall$ i.

Therefore, we have: $\mathbf{x}=\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ and $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$. Then, the hypothesis function can be written as:

$$\begin{aligned} h_\theta(x) &= \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 & \dots & \theta_n \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \\ &= \theta^T \cdot \mathbf{x} \end{aligned} \tag{3}$$

This is *Multivariate linear regression.*

4.2 Gradient Descent for Multiple Variables

4.2.1 Algorithm

Summary for Multivariables

1. **Hypothesis** $h_\theta(x) = \theta^T \cdot \mathbf{x}$
2. **Parameters** θ
3. **Cost Function** $J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$
4. **Goal** $\min_\theta J(\theta)$

Gradient Descent for Multiple Variables repeat until convergence{

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad \text{(for j=0, 1,\dots n)}$$

}

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(\mathbf{x}^{(\mathbf{i})}) - y^{(i)}) \cdot x_j^{(i)}$$

Note: $x_0^{(i)} = 1$, by definition.

4.2.2 Vectorized Implementation

One can work out the linear algebra and arrive at the following simplification using vectorized operations. The cost function J can be expressed as:

$$J(\theta) = \frac{1}{2m}(\mathbf{X}\theta - \mathbf{y})^T(\mathbf{X}\theta - \mathbf{y}) \tag{4}$$

The MATLAB implementation is as follows:

```
m = length(y); % calculate how many samples
J = 1/(2*m)*((X*theta-y).')*(X*theta-y);
```

Gradient descent can be vectorized in the form:

$$\theta = \theta - \frac{\alpha}{m} \cdot \mathbf{X}^T \cdot (\mathbf{X}\theta - \mathbf{y}) \tag{5}$$

The MATLAB implementation is as follows:

```
m = length(y); % number of training examples
for iter = 1:num_iters
    theta = theta - alpha/m* X.'*(X*theta -y);
```

4.3 Gradient Descent in Practise I: Feature Scaling

- Idea: ensure each featurre are on a similar scale
- Get every feature into approx. $-1 \leq x_i \leq 1(\sim$ order)
- **Mean Normalization**: Replace x_i with $\frac{x_i - \mu_i}{s_i}$, where μ_i and s_i are the sample mean and standard deviation, respectively.

4.4 Gradient Descent in Practise II: Learning Rate

- Ensure gradient descent is working: plot J_θ over each number of iteration (not over θ !)
- Example automatic convergence test: for sufficiently small α J_θ should decrease by less than 10^{-3} i one iteration.
- If α is too small, gradient descent can be slow to converge.
- If α is too large, gradient descent may not converge.
- To choose α, try 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1... (by 3x)

4.5 Features and Polynomial Regression

We can fit into different polynomials by choice, using multivariate regression. Recall

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

Let x_1 be x^1, x_2 be x^2, x_3 be x^3. Note we should still apply feature scaling to x_1, x_2, and x_3 individually!

4.6 Normal Equation

The normal equation provides a method to solve for θ analytically. For our data with m samples, n features, recall each sample can be written as:

$$\mathbf{x^{(i)}} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_j^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix}$$

We can construct a design matrix:

$$\mathbf{X} = \begin{bmatrix} — & (x^{(1)})^T & — \\ — & (x^{(2)})^T & — \\ — & (x^{(3)})^T & — \\ & \vdots & \\ — & (x_m^T & — \end{bmatrix} \tag{6}$$

Then θ can be found by the normal equation:

$$\theta = (\mathbf{X^T X})^{-1}\mathbf{Ty} \tag{7}$$

Normal equation is useful as no α is required to and we do not need to iterate. However, we do have to compute $(X^T X)^{-1}$, which can be computationally expensive when n is large. The complexity is $\mathrm{O}(n^3)$ for inverse operations. Gradient Descent is useful when n is large (many features).

4.7 Normal Equation and Non-invertibility

What if $(X^T X)^{-1}$ is non-invertible?

- Redundant features (linearly dependent), i.e. having same information in two different units.
- Too many features (i.e. m $\leq$ n). Delete some features or use regularization

5 Octave/MATLAB Tutorial

6 Logistic Regression

6.1 Classification

- Binary Classification: y $\in \{0, 1\}$, where 0 denotes the negative class; 1 denotes the positive class.

- Multi-class Classification:y $\in \{0, 1, \cdots, n\}$

We will be using binary classification:

- Linear regression is not suitable for classification: since $h_\theta(x)$ can output out of range, i.e. < 0 or > 1.
- We will use **logistic regression**, which ensures that the output $h_\theta(x)$ is between 0 and 1.

6.2 Hypothesis Representation

6.2.1 Logistic function

The idea is to have $0 \leq h_\theta(x) \leq 1$. Instead of the linear regression hypothesis: $h_\theta(x) = \theta^T x$, we will let:

$$h_\theta(x) = g(\theta^t x),$$

where

$$g(z) = \frac{1}{1 + e^{-z}}$$

g(z) is known as the **logistic function**, also known as the Sigmoid function. Figure 3 shows a plot of the logistic function, which ranges from 0 to 1.

which yields

$$h_\theta(x) = \frac{1}{1 + e^{-z}} \tag{8}$$

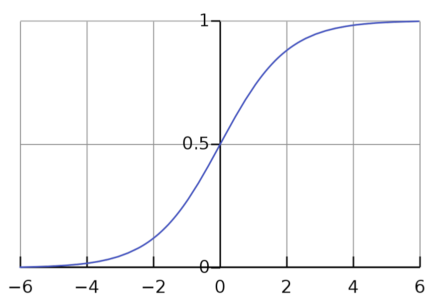

Figure 3: Sigmoid Function

6.2.2 Interpretation of Hypothesis Output

$h_\theta(x)$ = estimated probability that y= 1 on input x. For example, $h_\theta(x) = 0.7$ gives us a probability of 70% that are output is 1. From a probability theory point of view, one can express $h_\theta(x)$ as $P(y = 1 \mid x; \theta)$.

Note that since this is a probability and the total probability sums up to 1, and the real y can only be either 0 or 1:

$$P(y = 0 \mid x; \theta) + P(y = 1 \mid x; \theta) = 1$$

6.3 Decision Boundary

Recall so far we have $h_\theta(x) = g(\theta^T x)$ and $g(z) = \frac{1}{1+exp(-z)}$. Suppose we set $h_\theta(x) = 0.5$ to be our determining factor for whether y= 0 or y =1. Note that from 3, one can observe that $h_\theta(x) = 0.5$ corresponds to $\theta^T x = 0$, which is the **decision boundary**. The decision boundary is the equation which separates the different classes on a plot. There are linear and non-linear decision boundaries.

6.4 Cost function

Previously, we had $J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$, where $Cost(h_\theta(x), y) = \frac{1}{2}(h_\theta(x) - y)^2$. Now, the definition of the hypothesis h_θ has changed to $\frac{1}{1+exp(-\theta^T x}$, as a result the cost function is now non-convex.

Logistic Regression Cost Function
Therefore, a new cost function definition is needed. We propose:

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Note that Cost=0, if y=1, $h_\theta(x) = 1$; but as $h_\theta(x) \to 0$, then $Cost \to \infty$. This proposition captures the intuition that if $h_\theta(x) = 0$, predict $P(y = 1 \mid x; \theta)$, but y ends up being 1, we will penalize the learning algorithm by very large cost.

6.5 Simplified Cost Function and Gradient Descent

Since y can only be either 0 or 1, we can simplify the cost function.

$$\begin{aligned} J(\theta) &= \frac{1}{m}\sum_{i=1}^{m} Cost\,(\,h_\theta(x^{(i)})\,,\,y^{(i)}) \\ &= \frac{-1}{m}\,[\,\sum_{i=1}^{m} y^{(i)}\,log\,h_\theta(x^{(i)})\;+\;(1-y^{(i)})\,log\,(\,1-h_\theta(x^{(i)})\,)\,] \end{aligned} \tag{9}$$

Equation 9 is based on Maximum Likelihood Estimation.
A vectorized implementation is, for a design matrix $\mathbf{X}$:

$$\boxed{h = g(\mathbf{X}\theta)} \tag{10}$$

$$\boxed{J(\theta) = \frac{1}{m}(-y^T log(h) - (1-y)^T log(1-h))} \tag{11}$$

For gradient descent,we would want to $\min_\theta J(\theta)$:
Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

}

We can compute the partial derivative of $J(\theta)$, which is identical to that of linear regression:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

However, in this case the hypothesis function $h_\theta(x) = \frac{1}{1+exp(-\theta^T x}$ has changed!

A vectorized implementation for this is:

$$\boxed{\theta := \theta - \frac{\alpha}{m} \mathbf{X}^T (g(\mathbf{X}\theta) - \mathbf{y}))} \tag{12}$$

6.6 Advanced Optimization

6.6.1 Taking a Step Back

If we take a step back, and consider essentially what tasks we are performing. We need to compute two things:

1. $J(\theta)$
2. $\frac{\partial}{\partial \theta_j} J(\theta)$

6.6.2 Optimization Algorithm

There exists other more sophisticated and faster ways to optimize θ instead of gradient descent; they often do not involve selecting learning rate α and are more efficient. However, these algorithms are harder to code by hand. It is suggested that we use libraries for such algorithms.

We can write a single function that returns both $J(\theta)$ (jVal) and $\frac{\partial}{\partial \theta_j} J(\theta)$ (gradient):

```
function [jVal, gradient] = costFunction(theta)
    jVal = [...code to compute J(theta)...];
    gradient = [...code to compute derivative of J(theta)
        ...];
end
```

Then we can use octave's "fminunc()" optimization algorithm along with the "optimset()" function that creates an object containing the options we want to send to "fminunc()".

```
options = optimset('GradObj', 'on', 'MaxIter', 100);
initialTheta = zeros(2,1);
[optTheta, functionVal, exitFlag] = fminunc(
    @costFunction, initialTheta, options);
```

We then give to the function "fminunc()" our cost function, our initial vector of theta values, and the "options" object that we created beforehand.

6.7 Multiclass Classification: One-vs-All

Now, let's extend the binary classification of data to multi-classes, i.e expanding our definition of y s.t. y={0, 1, . . . , n}. We will divide out problem into n+1 (0. . . n) binary classification problems. In each problem, we predict the probability that y is a memember of one of our class. We train a logistic regression classifier $h_\theta^{(i)}(x)$ $\forall$ i to predict y = i:

$$h_\theta^{(i)}(x) = P(y = i \mid x; \theta) \tag{13}$$

Figure 4 shows an example of the procedure of classifying three classes. We choose one class and then lump all the others into a single second class (hence the name One-vs-All). We apply the binary logisic regression repeatedly and use the hypothesis that returns the highest value.

$$prediction = \max_i \left(h_\theta^{(i)}(x) \right) \tag{14}$$

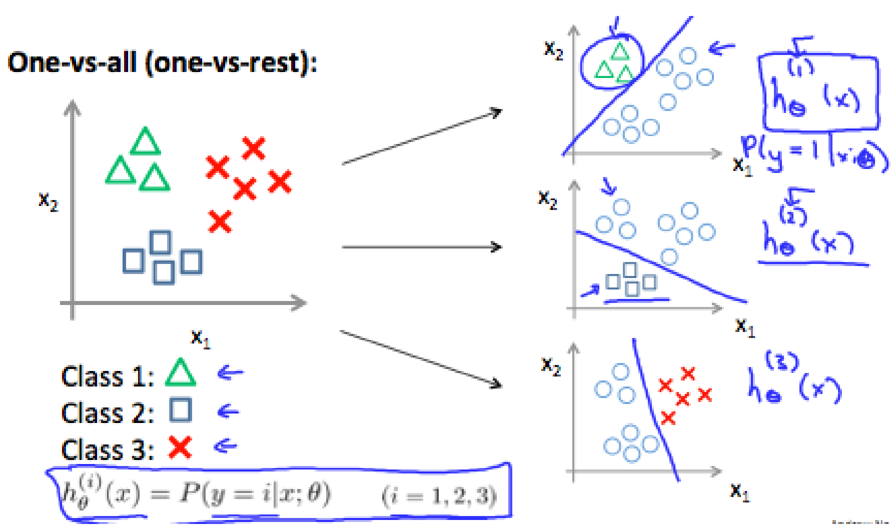

Figure 4: Example of Multiclass Logistic Regression